Computing and Informatics, Vol. 31, 2012, 1511–1531

DECOMPOSABLE NAIVE BAYES CLASSIFIER FOR PARTITIONED DATA

Ahmed M. Khedr

Computer Science Department, Faculty of Sciences
Sharjah University, Sharjah, UAE
&
Mathematics Department, Faculty of Sciences
Zagazig University, Zagazig, Egypt
e-mail: akhedr@sharjah.ac.ae, amkhedr@zu.edu.eg

Communicated by Jozef Kolemen

Abstract. Most learning algorithms are designed to work on a single dataset. However, with the growth of networks, data is increasingly distributed over many databases in many different geographical sites. These databases cannot be moved to other network sites due to security, size, privacy, or data ownership consideration. In this paper, we propose two decomposable versions of Naive Bayes Classifier for horizontally and vertically partitioned data. The goal of our algorithms is to achieve the learning objectives for any data distribution encountered across the network by exchanging minimum local summaries among the participating sites.

Keywords: Agents, decomposable algorithms, naive bayes classifier, vertical and horizontal partitions

Mathematics Subject Classification 2010: 68U99

1 INTRODUCTION

Classification is a predictive modeling task with the specific aim of predicting the value of a single nominal variable based on the known values of other variables. There are many practical situations in which classification is of immense use. Examples include: providing a diagnosis for a medical patient based on a set of test

results, estimating the probability of purchase of a given item given the other items purchased, and others.

Though there are some organizations which single-handedly collect a lot of data on their own, often large correlated data is collected over many sites. It is possible that several organizations collect similar data about different people (horizontal partitioning of data). Examples include: banks collecting credit card information for their customers or supermarkets collecting transaction information for their clients. On the other hand, different organizations may collect different information about the same set of people (vertical partitioning of data). Examples of this include: hospitals and insurance companies collecting information or producer/consumer industrial concerns collecting data which can be jointly linked.

The Naive Bayes classifier (NB-classifier) is a simple but efficient baseline classifier. NB-classifier is based on a Bayesian formulation of the classification problem which uses the simplifying assumption of attribute independence. It is simple to implement and use while giving surprisingly good results. Thus, preliminary evaluation is carried out using the NB-classifier to serve both as a baseline and to decide whether more sophisticated solutions are required.

The problem of secure distributed classification is an important one. The goal is to have a simple, efficient and privacy-preserving classifier. The sites have agreed to work together so that a user at any site can access data anywhere in the network exactly as if the data were all stored at the user's own site; and all the data can be simultaneously accessed. The site that initiated request is called the *Learner* site (the site to which a user is directly connected), and any other site is called *remote* site (any site accessed by that user). While a distributed environment enables increased access to a large amount of data across a network, it must also hide the location of the data and the complexity of accessing it across the network. The main limitations in these situations are that databases cannot move to a common site due to size, security and privacy consideration; cannot update local databases; and cannot send actual data tuples [1, 2, 3, 4, 5, 6, 7, 8].

In a *distributed learning* scenario, the data set is assumed to be physically distributed across multiple autonomous sites and the learner's task is to acquire useful knowledge from this data. Learning can be accomplished by an agent that visits the different sites to gather the information necessary for generating knowledge (e.g., in the form of pattern classification rules) by processing the data where it is stored. Alternatively, the different sites can transmit the information necessary to the learning agent situated at the Learner site. In either case, we prohibit transport of raw data among different sites. Consequently, the learner has to rely on information (e.g., number of instances that match some criteria of interest) extracted from the sites.

We present a methodology for constructing a NB-classifier across multiple distributed databases. This methodology consists of a general model for decomposable NB-classification and a set of algorithms for realizing this model. Our approach to learn from distributed data sets involves identifying the information requirements of existing learning algorithms, and designing efficient means of providing the neces-

sary information to the learner, while avoiding the need to transmit large quantities of data. This decomposition of the learning task into *information extraction* and *hypothesis generation* components offers a general approach to adapting the existing learning algorithms to work in the distributed setting. In this model of distributed learning, only the information extraction component has to effectively cope with the distributed nature of the data.

The rest of the paper is organized as follows: Section 2 describes the related work in this area. Section 3 briefly describes integration of distributed data. Section 4 presents the NB-classifier from horizontally and vertically distributed data. Section 5 presents the simulation results. In Section 6, we discuss the advantages and security considerations of our algorithms. Finally, Section 7 concludes the paper.

2 RELATED RESEARCH

Many practical knowledge discovery tasks present several new challenges in machine learning. The data repositories in such applications tend to be very large, physically distributed, and autonomously managed. Learning from databases is a great practical value in a variety of application domains and NB-classifier is an old and well-known type of classifiers. Most of the learning algorithms in the literature assume that all the relevant data is available in a single computer site [9].

In the context of database research, some work has been done towards learning from distributed databases for example [10, 11, 12, 13, 14, 15]. Most of the existent algorithms work for horizontal data distributions, with a few exceptions. [15] proposed methods for distributing a large centralized data set to multiple processors to exploit parallel processing to speed up learning. [12, 14] surveyed several methods that exploit parallel processing for scaling up data mining algorithms to work with large data sets. In contrast, the focus of our work is on learning classifiers from a set of autonomous distributed data sources.

Several distributed learning algorithms have their roots in ensemble methods, [10, 11, 13] proposed an ensemble of classifiers approach to learn from horizontally fragmented distributed data which essentially involves learning separate classifiers from each data set and combining them typically using a weighted voting scheme. This requires gathering a subset of data from each of the data sources at a Learner site to determine the weights to be assigned to the individual hypotheses (or shipping the ensemble of classifiers and associated weights to the individual data sources where they can be executed on local data to set the weights). In contrast, our approach is applicable even in scenarios which preclude transmission of data or execution of user-supplied code at the individual data sources but allow transmission of minimal sufficient statistics needed by the learning algorithm. The second potential drawback of the ensemble of classifiers approach to learn from distributed data is that the resulting ensemble of classifiers is typically much harder to comprehend than a single classifier. The third important limitation of the ensemble classifier approach to learn from distributed data is the lack of strong guarantees concerning

accuracy of the resulting hypothesis relative to that obtained in the centralized setting. In contrast, we prove that our resulting hypothesis is similar to the hypothesis obtained in the centralized setting.

The task of learning from relational data has received significant attention in the literature in the last few years. One of the first approaches to relational learning was based on Inductive Logic Programming (ILP) [16]. ILP is a broad field which evolved from the development of algorithms for the synthesis of logic programs from examples and background knowledge to the development of algorithms for classification, regression, clustering, and association analysis [17].

In [18], the authors gave a good explanation why a NB-classifier works surprisingly well despite its strong independence assumption. However, since those assumptions rarely ever hold in real applications, it is interesting to explore networks beyond a NB-classifier. Although the reasons explaining the competitiveness of NB-classifier remain unclear, several studies have revealed useful information; examples include studies about the conditions for its optimality [18]; its geometric properties [19]; and how the product distribution implied by the independence assumption compares to most other joint distributions with the same set of marginals [20].

The work in [21] provided privacy-preserving solutions for mining a NB-classifier across a database horizontally partitioned into a small number of partitions. The author used secure summation and logarithm techniques to create a privacy preserving NB-classifier for horizontally partitioned data. Our approach is based on a general strategy for transforming traditional machine learning algorithms into distributed learning algorithms based on the decomposition of the learning task into hypothesis generation and information extraction components; formally defines the information required for generating the hypothesis (sufficient statistics); and shows how to gather the sufficient statistics from distributed data sources. We decompose the counting process in such a way that various *local* count requests can be sent to sites of individual $D_k s$ and the responses can then be composed to reconstruct the *total* count for the examples in implicit training set.

The work in [22] provided privacy-preserving solutions for mining a NB-classifier across a database vertically distributed data mining scenario when different sites contain different attributes for a common set of entities (special case of distributed data). Security requirements to be met in this algorithm demand that a site can know only its own attribute-value pairs when they become parts of the global one. Our algorithm does not assume any conditions on the data partition at any one of the participating sites.

Central to our approach is a clear separation of concerns between hypothesis construction and extraction of sufficient statistics from data, making it possible to explore the use of sophisticated techniques for query optimization that yield optimal plans for gathering sufficient statistics from distributed data sources under a specified set of constraints describing the query capabilities and operations permitted by the data sources.

3 INTEGRATION OF DISTRIBUTED DATA

In a distributed setting, a dataset $\mathcal{D}$ implicitly defined in n explicit databases D_i located at n different sites. We model databases D_i at the i^{th} site, by a *relation* containing a number of tuples. Each D_i contains set of attributes represented by X_i. For any two databases D_i and D_j, the corresponding sets X_i and X_j may have a set of shared attributes given by S_{ij}. Since an arbitrary number of independent, already existing, databases may be consulted for a computation, we cannot assume any data normalization to have been performed for their schemas.

The implicit data set $\mathcal{D}$ with which the computation is to be performed is a subset of the set of tuples generated by a *Join* operation performed on all D_i. However, the tuples of $\mathcal{D}$ cannot be made explicit at any one network site because entire databases, D_i's, cannot be moved to other sites. The tuples of $\mathcal{D}$, therefore, must remain implicitly specified. This inability to make explicit the tuples of $\mathcal{D}$ is the main problem addressed in the generalized decomposition of global algorithms. To facilitate computations with implicitly specified $\mathcal{D}$, we define a set S that is the union of all the attribute intersection sets S_{ij}, that is,

$$S = \bigcup_{i \neq j} S_{ij}. \tag{1}$$

The set S contains the names of all those attributes that occur in more than one D_i. We define a relation *Shareds* which contains all possible tuples that can be enumerated for the attributes in the set S.

3.1 Nature of Data Distribution

The dataset $\mathcal{D}$ consists of a set of tuples where each tuple stores the values of relevant attributes. The distributed nature of such a dataset can lead to two common types of data partition: *horizontal partition* wherein subsets of data tuples are stored at different sites; and *vertical partition* wherein subtuples of data tuples are stored at different sites. Assume that a dataset $\mathcal{D}$ is distributed among n sites containing databases $D_1, D_2, \ldots, D_n$. The individual databases D_i, together, constitute the implicit global dataset $\mathcal{D}$.

Horizontally Partitioned Datasets: In this scenario each component D_i of D_1, D_2, …, D_n contains the same attribute set X_i, but a different set of data tuples. The set of *Shared* attributes S is the same as X_i, for each database. The union of all databases D_i constitutes the complete dataset $\mathcal{D}$, i.e., $D_1 \cup D_2 \cup \ldots \cup D_n = \mathcal{D}$.

Vertically Partitioned Datasets: In this scenario, each component D_i consists of tuples formed with a different set of attributes but each with those of some other databases, D_j, $j \neq i$. Each D_i may also contain some attributes that are unique to the local site and are not *Shared* with a database at any other site.

In effect, each D_i is a projection of an implicit global $\mathcal{D}$. Vertically fragmented datasets are of more interest because they provide an opportunity to share knowledge across the participating sites.

As shown in Figure 1 each D_i is represented by an agent $Agent_i$ that communicates with similar agents at other sites to exchange simple computational summaries. The algorithms discussed here can be seen to reside with these agents and any one of these agents is capable of initiating and completing a computational task by exchanging summaries with agents at other sites.

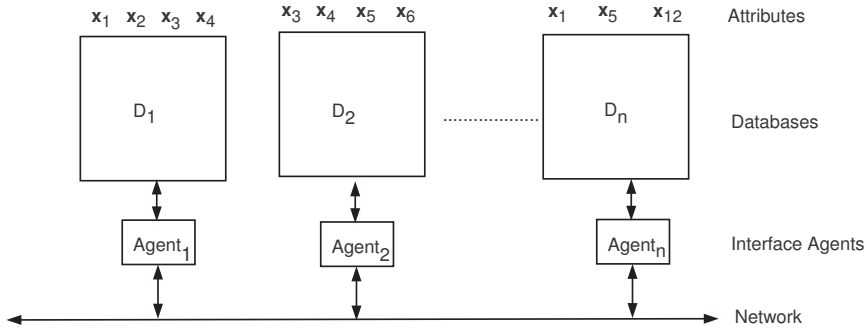

Fig. 1. Distributed data/knowledge sources

3.2 Agent's Decomposition Task

The objective of an agent is to perform the global computation by communicating with other similar agents at other sites; and each agent performing some computation with its local database. Each agent should be able to decompose the global computation into local computations – in the context of and as constrained by the sharing of attributes across the participating agents – and perform its local part with its own data. Each agent $Agent_i$ in Figure 1 represents a D_i and communicates with similar agents at other nodes to exchange the results of its local computations. The decomposition methodologies discussed here can be seen to reside with each individual agent; and each agent is also capable of initiating and completing an instance of a global computation by either exchanging local results with other agents, stationary at their respective sites, or by launching a mobile agent that visits other network sites. In the case of a mobile agent the decomposition tools and knowledge reside with the mobile agent.

Let us say a result $\mathcal{R}$ is to be obtained by applying a function $\mathcal{F}$ to the implicit dataset $\mathcal{D}$. That is:

$$\mathcal{R} = \mathcal{F}(\mathcal{D}). \tag{2}$$

When the global computation is to find a count in distributed data components, the value of $\mathcal{R}$ is the count across the global data; $\mathcal{D}$ is the database containing the data;

and $\mathcal{F}$ corresponds to the implementation of an algorithm for inducing $\mathcal{R}$, from $\mathcal{D}$. Distributed databases used by the agents cannot make explicit the tuples of $\mathcal{D}$, which remain implicit in terms of the explicitly known components $D_1, D_2, \ldots, D_n$. The set S of *Shared* attributes determines what explicit $\mathcal{D}$ would be generated by the individual data components. An implementation of $\mathcal{F}$ in Equation (2) above, for some S, can be engineered by a functionally equivalent formulation given as:

$$\mathcal{R}(S) = H[h_1(D_1, S), h_2(D_2, S), \ldots, h_n(D_n, S)]. \tag{3}$$

That is, a local computation $h_i(D_i, S)$ is performed by agent $Agent_i$ using the database D_i and the knowledge about the attributes *Shared* among all the data sites (S). The results of these local computations are aggregated by an agent using the operation H. However, it may not be possible to decompose a complex computation algorithm into local computations and an aggregator. In this case, we can decompose smaller computational primitive steps of such a complete algorithm and the agent keeps track of the control aspects of sequencing various steps of such an algorithm.

The number and nature and h_i operators and the nature of H would vary with the participating D_is and the set of attributes S among them. Hence, a different set of h-operators would need to be generated by the agent for each new instance of D_i's and S.

Figure 2 shows the process by which the agent would compute $\mathcal{R}$ from the D_is. The component operators of a decomposition (H and h_is), therefore, need to be dynamically determined by the agent for each instance of $\mathcal{F}(\mathcal{D})$ depending on the participating nodes, the attributes contained in their native databases, and the sharing pattern of attributes.

3.3 Software Agents

In our work, we consider *Agent* as "a piece of software which performs a given task using information gleaned from its environment to act in a suitable manner so as to complete the task successfully". An agent should be able to adapt itself based on changes occurring in its environment, so that a change in circumstances will still yield the intended result. Agents are used to represent actors in a cooperative effort, and give the users of the agent system support for doing efficient negotiation, and exchange of information.

Agents may be classified by their *mobility*, i.e. by their ability to move around some network. This yields the classes of *stationary* agents that stay at their respective data sites; or *mobile* agents that move from one site to the other.

Stationary Agent executes only on the site where it begins execution. If it needs information that is not on that site, or needs to interact with other similar agents at other sites for exchanging (sending/receiving) simple computational summaries to perform the global computations, it uses message passing as a communication mechanism.

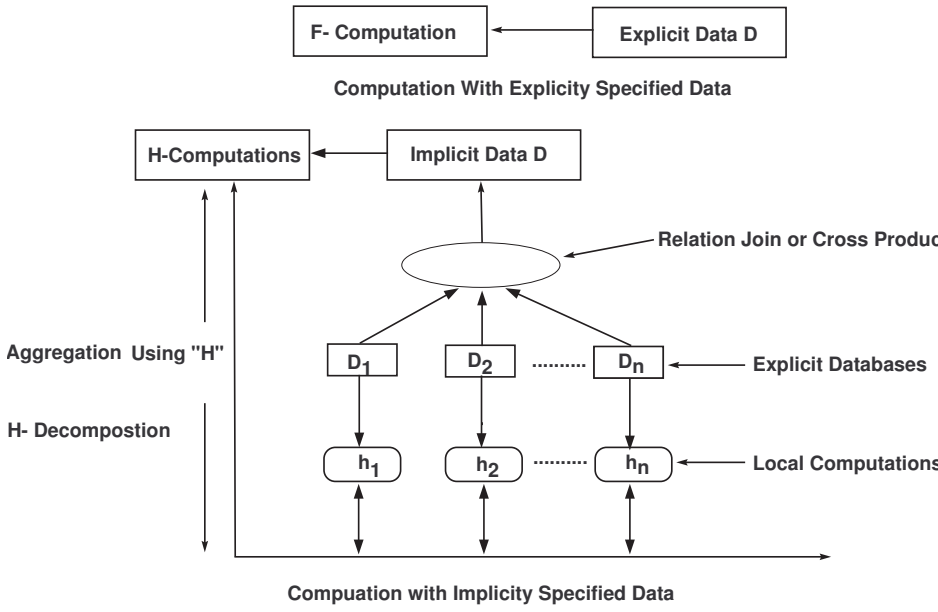

Fig. 2. Computations in explicit vs. implicit data spaces

Mobile Agent is not bound to the system where it begins execution. The mobile agent is free to migrate during execution from one site to another where it can resume its execution in order for perform local computations at each site that it visits, and at the end compose the gathered results for performing the global computations.

3.4 Cost Models for Algorithmic Complexity

Traditionally, the complexity of algorithms has been measured in terms of the CPU time and the required memory. This cost model is well-suited for computations on a single computer and the closely-coupled processors model. When a number of loosely networked nodes are involved in a cooperative computation the communication cost becomes the overwhelmingly dominant component of the total cost. Complexity for distributed query processing in databases has been discussed in [26]. In our experience with the design and analysis of decomposable network algorithms, we have found that each step of the algorithm must exchange a number of messages for evaluating the various quantitative values. Here and in other similar work [1, 2, 3, 4, 5, 6, 7], we have used cost models involving the number of messages exchanged and reflecting the efficiency of decomposition carried out by the network algorithm.

4 LEARNING NB-CLASSIFIER FROM DISTRIBUTED DATA

Traditionally, the NB-classifier applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function $f(x)$ can take on any value from some finite set V [23, 24, 25]. A set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values $\langle a_1, a_2, \ldots, a_m \rangle$. The learner is asked to predict the target value, or classification, for this new instance. The Bayesian approach to classify the new instance is to assign the most probable target value, v_{MAP}, given the attribute values $\langle a_1, a_2, \ldots, a_m \rangle$ that describe the instance.

$$v_{MAP} = \text{argmax}_{v_j \in V}(P(v_j | a_1, a_2, \ldots, a_m)). \tag{4}$$

Using Bayes theorem,

$$v_{MAP} = \text{argmax}_{v_j \in V}(P(a_1, a_2, \ldots, a_m) P(vj)). \tag{5}$$

The NB-classifier makes the further simplifying assumption that the attribute values are conditionally independent given the target value. Therefore,

$$v_{NB} = \text{argmax}_{v_j \in V} P(v_j) \prod_{i=1}^{m} P(a_i | v_j), \tag{6}$$

where v_{NB} denotes the target value output by the NB-classifier. The conditional probabilities $P(a_i|vj)$ need to be estimated from the training set. The prior probabilities $P(v_j)$ also need to be fixed in some fashion (typically by simply counting the frequencies from the training set). The probabilities for differing hypotheses (classes) can also be computed by normalizing the values received for each hypothesis (class). Probabilities are computed differently for nominal and numeric attributes.

4.1 Sufficient Statistics for NB-Classifiers

In NB-classifier, a hypothesis is built during the learning phase, and is used during the classification phase to classify new instances. The set of probabilities $P(v_j)$ and $P(a_i|v_j)$, representing the hypothesis, can be computed based on the following counts:

- N_t is the total number of training examples.
- N_{v_j} is the number of training examples in class v_j.
- N_{ij} is the number of training examples in class v_j and has the attribute value a_i.

These counts represent the sufficient statistics for the hypothesis build during the learning phase of NB-classifier. In our distribution version of NB-classifier, we will show how the minimal sufficient statistics can be computed, when data is horizontally and vertically distributed.

4.2 Decomposable NB-Classifier for Horizontally Partitioned Data

In the horizontally distributed setting, we compute the minimal sufficient statistics by computing the following counts at every participating site k and then ship the results to the Learner:

- The total number of training examples (N_t^k).
- The number of training examples in class v_j, ($N_{v_j}^k$) .
- The number of training examples in class v_j that have the attribute value a_i, $\left(N_{ij}^k\right)$.

Since in horizontally $D = D_1 \bigcup D_2 \bigcup \ldots \bigcup D_n$, hence $|D| = |D_1| + |D_2| + \ldots + |D_n|$. Therefore, the global counts are obtained at the Learner site by adding up local counts. The pseudocode for horizontally distributed data is shown below:

Learning Phase: Given a new instance $x =< a_1, a_2, \ldots, a_m >$ to be classified.

Local Computation: the following code will be executed at every participating site D_k

1. Compute the count of training examples N_t^k
2. for every class label v_j do
   (a) Compute the count of training examples in class v_j, $N_{v_j}^k$
   (b) for every attribute value a_i do
      i Compute the count of training examples in class v_j and has the attribute value a_i, N_{ij}^k
   (c) Ship back all computed counts to the Learner site

Global Computation: At the Learner site and from the shipped data, compute the following:

$N_t = \sum_{k=1}^{n} N^k$, $N_{v_j} = \sum_{k=1}^{n} N_j^k$, and $N_{ij} = \sum_{k=1}^{n} N_{ij}^k$ $P(v_j) = \frac{N_{v_j}}{N_t}$, and $P(a_i|v_j) = \frac{N_{ij}}{N_{v_j}}$.

Classification Phase: For the instance $x =< a_1, a_2, \ldots, a_m >$ compute

$$v_{NB} = \text{argmax}_{v_j \in V} P(v_j) \prod_{i=1}^{m} P(a_i|v_j). \tag{7}$$

4.2.1 Complexity and Analysis

Stationary Agent Case

Let us say, there are n relations, $D_1, D_2, \ldots, D_n$ residing at n different network sites.

Cost Model 1: In this cost model, the number of required exchanged messages will be $3 * n$, where the number of messages required to compute N_t is n, the number of messages required to compute N_{v_j} is n, and the number of messages required to compute N_{ij} is n. The result shows that the number of messages that need to be exchanged among the sites is not dependent on the size of the database at each site. This is significant because it shows that as the sizes of the individual databases grow, the communication complexity of our algorithm would remain unaffected.

Cost Model 2: In this cost model, all the summaries are sent in one message. Each count would require to exchange only one message. Thus, the total number of messages will be 3 exchanged messages.

The trade-off between the two approaches is that the first one may be considered more secure for transmission over a network because each message contains only little information about the participating databases. The second alternative requires very few messages but each message contains more information about each database.

Mobile Agent Case

The mobile agent has the relation *Shareds* stored in it. During a visit to a data site, it computes all local computations for that site. The local results for all counts can be gathered during a single visit to a site. Thus, the mobile agent can compute all probabilities for each new instance x by visiting each site only once (n hops) and then aggregate the local results.

Assertion 1. The algorithm for learning NB-classifiers from horizontally distributed data returns the same results with respect to the algorithm for learning NB-classifier from centralized data.

Proof. In horizontally distributed data, the training examples can be reconstructed from $D_1, D_2, \ldots, D_n$ by taking the union of these subsets, i.e., $D = D_1 \bigcup D_2 \bigcup \ldots \bigcup D_n$, hence $|D| = |D_1| + |D_2| + \ldots + |D_n|$. Therefore, the global counts are obtained at the Learner site by adding up the local counts, i.e., we obtain the same numbers as if we brought all the data together and compute the counts globally. □

4.3 Decomposable NB-Classifier for Vertically Partitioned Data

Here, we assume that the training examples are in the vertically distributed form. The algorithm for learning NB-classifier from vertically distributed data is shown in the following:

Learning Phase Given a new instance x to be classified, the following steps will be executed at the Learner site:

1. Call Count-Computing( ) to compute the total number of training examples N_t.

2. Call Count-ComputingN_{v_j}( ) to compute the count of training examples, N_{v_j}, in class v_j.
3. Call Count-ComputingN_{ij}( ) to compute the count of the training examples, N_{ij}, in class v_j that have attribute value a_i.
4. Compute $P(v_j) = \frac{N_{v_j}}{N_t}$, and $P(a_i|v_j) = \frac{N_{ij}}{N_{v_j}}$.

Classification Phase: For the instance $x = \langle a_1, a_2, \ldots, a_m \rangle$ return

$$v_{NB} = \text{argmax}_{v_j \in V} P(v_j) \prod_{i=1}^{m} P(a_i|v_j). \tag{8}$$

In the following, we introduce the Count-Computing procedures for computing minimal sufficient statistics from vertically distributed databases.

4.3.1 Count-Computing Procedure

When the training examples of a complete training set $\mathcal{D}$ are explicitly available in a relation then the count of all its tuples can be obtained easily. For the case of implicitly defined $\mathcal{D}$, we can decompose the counting process in such a way that various *local* count requests can be sent to sites of individual D_ks and the responses can then be composed to reconstruct the *total* count for the examples in implicit training set. The decomposition for obtaining the count N_t is as follows:

$$N_t = \sum_{l} \left(\prod_{k=1}^{n} N(D_k)_{cond_l} \right), \tag{9}$$

where the subscript $cond_l$ specifies a condition composed from the attribute value pairs of the l^{th} tuple of the relation *Shareds*, n is the number of participating databases (D_ks), and $(N(D_k)_{cond_l})$ is the count in relation D_k of those training examples that satisfy the condition $cond_l$. Each term in the product is the count of tuples which satisfy condition $cond_l$ in a D_k. The resulting product gives the number of distinct tuples that would be contributed to the implicit *join* of all the D_ks for the condition specified by $cond_l$. The summations in the above expression amount to selecting each tuple of *Shareds* as $cond_l$ and then summing the product terms obtained for each tuple. This expression, therefore, simulates the effect of a *join* operation performed on all the n databases without explicitly enumerating the tuples. A desirable aspect of the above decomposition of N_t is that each local computation $N(D_k)_{cond_l}$ can be translated into an SQL query: *Select count (*) where* $cond_l$ and sent to the site containing database D_k. The pseudocode for computing N_t is shown below:

Local Computation: The following code will be executed at every local data site D_k

1. for every *Shared* tuple l do
   (a) Compute $N(D_k)_{cond_l}$, and ship the results back to the Learner site
2. end for

Global Computation: The following code will be executed at the Learner site

1. for every *Shared* tuple l, compute the total number of training examples that satisfy $cond_l$ from the following relation:

$$N_l = \prod_{k=1}^{n} N(D_k)_{cond_l}. \tag{10}$$

2. Compute the total number of examples from the following relation:

$$N_t = \sum_l N_l = \sum_l \left(\prod_{k=1}^{n} N(D_k)_{cond_l} \right). \tag{11}$$

4.3.2 Count-Computing N_{v_j} Procedure

We can easily extend the above decomposition for count to the counts of only those tuples that satisfy certain condition ($class = v_j$) by simply changing $cond_l$ in Equation (9) to $cond_l.and.class = v_j$; then Equation (9) will be

$$N_{v_j} = \sum_l \left(\prod_{k=1}^{n} N(D_k)_{cond_l.and.class=c_j} \right). \tag{12}$$

4.3.3 Count-ComputingN_{ij} Procedure

We can easily determine the number of training examples in class v_j and have attribute value a_i by simply changing $cond_l$ in Equation (9) to ($cond_l.and.class = v_j.and.attr = a_i$); so the formula to compute N_{ij} will be

$$N_{ij} = \sum_l \left(\prod_{k=1}^{n} N(D_k)_{cond_l.and.class=v_j.and.attr=a_i} \right). \tag{13}$$

4.3.4 Complexity and Analysis

Stationary Agent Case

We give below an expression for the number of messages that need to be exchanged for dealing with the implicit number of tuples. Let us say there are:

- n relations, $D_1, D_2, \ldots, D_n$ residing at n different network sites,
- r tuples in the relation *Shareds*.

Cost Model 1: The number of messages needed will be the sum of the number of messages required to compute N_t, the number of messages required to compute N_{v_j}, and the number of messages required to compute N_{ij}, each of which will take $n*r$ messages. Thus, the total number of messages exchanged will be:

$$Total\ Exchanged\ Messages = 3n * r. \tag{14}$$

Cost Model 2: In this cost model, values corresponding to all tuples $cond_j$ of S will be sent in one request and then receive the summaries in one message. This reduces the number of messages exchanged to n, the same as the number of participating sites. Thus, the total number of messages exchanged will be:

$$Total\ Exchanged\ Messages = 3 * n. \tag{15}$$

Mobile Agent Case

This agent has the relation *Shareds* stored in it. During a visit to a data site, it computes all local computations for that site. The local results for computing all the counts can be gathered during a single visit to a site. Thus, the mobile agent can compute all probabilities for each new instance x by visiting each site only once (n hops) and then aggregating the local results.

Assertion 2. The algorithm for learning NB-classifiers from vertically distributed data return the same results with respect to the algorithm for learning NB-classifiers from centralized data.

Proof. It is obvious that the counts N_t, N_{v_j}, N_{ij} computed in the distributed case are the same as the counts computed in the centralized case. Then the set of probabilities $P(v_j)$, $P(a_i \setminus v_j)$ are identical. i.e., we obtain the same results as if we brought all the data together and compute these probabilities globally. □

5 SIMULATION RESULTS

We have performed a number of tests to demonstrate that the NB-classifier can be run in a distributed knowledge environment without moving all the databases to a single site. These tests have been carried out on a network of workstations connected by a LAN and tested against different sizes of databases, different number of shared tuples, and different number of local sites. We have implemented the algorithm using Java, RMI (Remote Method Invocation), and JDBC (Java Database Connectivity) to interface with the databases.

In the first test, we demonstrate how the elapsed time and the number of exchanged messages varies with the number of local sites. The number of the local sites varies between 2 and 6 with increment of 1. Figure 3 shows how the elapsed time to compute NB-classifier in an implicit database $\mathcal{D}$ changes with the number of local sites. It can be seen easily that the elapsed time increases with the number of local sites. Also, Figure 4 shows that the number of exchanged messages increases with the number of local sites.

In the second test, we demonstrate how the elapsed time and the number of exchanged messages varies with the average number of shared tuples between local databases. The number of shared values varies between 5 and 25 with increment of 5. Figure 5 shows the elapsed time to compute NB-classifier in an implicit database $\mathcal{D}$. The figure shows that the elapsed time increases as the number of shared values

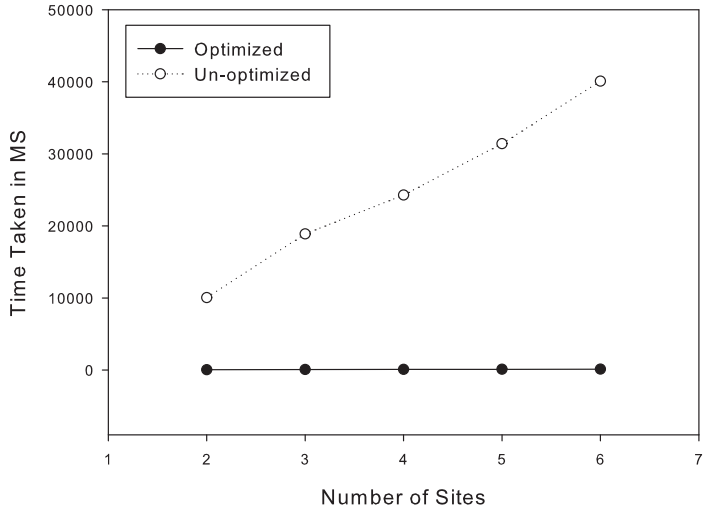

Fig. 3. Elapsed time to run NB-classifier on vertically partitioned data (different local sites)

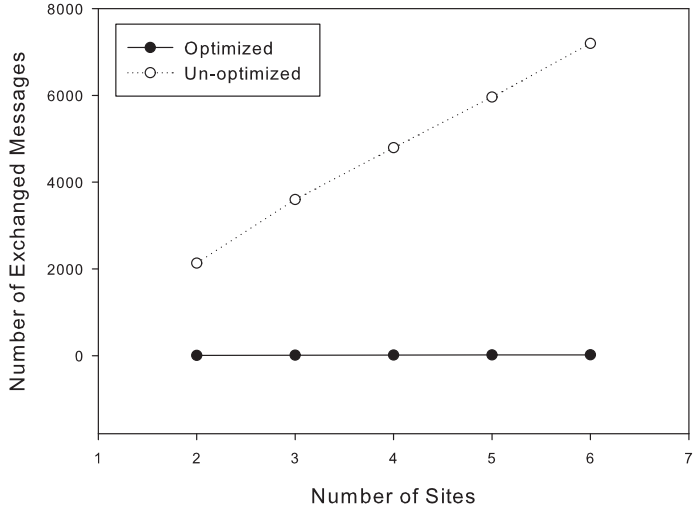

Fig. 4. Number of messages exchanged to run NB-classifier on vertically partitioned data (different local sites)

increases. Also, Figure 6 shows that the number of exchanged messages increases as the number of shared values increases.

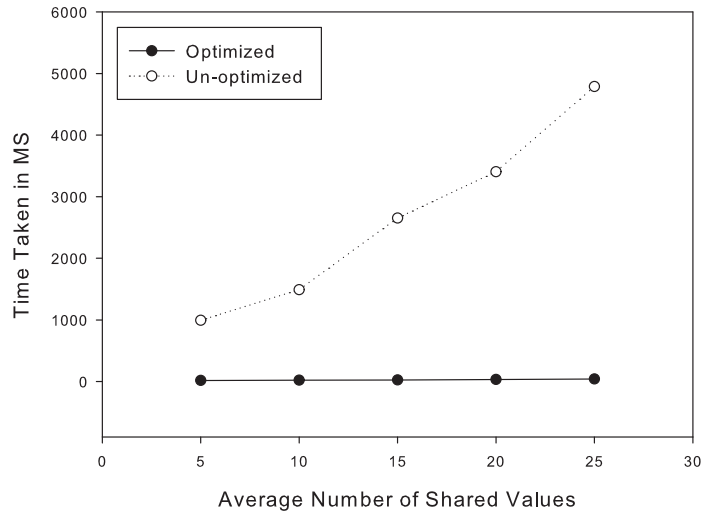

Fig. 5. Elapsed time to run NB-classifier on vertically partitioned data (different number of Shared tuples)

In the last test, we demonstrate how the elapsed time and the number of exchanged messages vary with the number of tuples in the database. Figure 7 shows the change between elapsed time to compute NB-classifier in an implicit database $\mathcal{D}$ and the number of tuples in the database. It shows that the time taken to compute NB-classifier in an implicit database $\mathcal{D}$ changes with the size of the individual databases. As we can see, when we exchange one summary per message, the time taken to run the NB-classifier varies as the size of the database increases. However, when we use the optimized method the time taken to run the NB-classifier reduces considerably and depends on the number of participating nodes.

Figure 8 shows the change between the number of exchanged messages and the number of tuples in the database. It shows how the number of messages exchanged between the Learner site and the remote sites varies with the number of tuples in the database. It can be seen easily that the number of messages exchanged increases with the size of the database when we send one summary per message. However, in the optimized version when we receive all the summaries in a single message, the number of messages exchanged was a constant depending upon the total number of participating nodes. The result validates the expression for the total number of messages exchanged as given above.

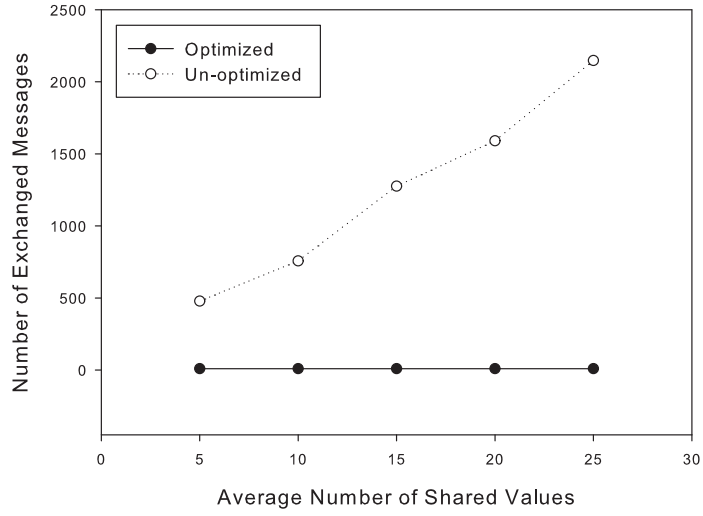

Fig. 6. Number of messages exchanged to run NB-classifier on vertically partitioned data (different number of Shared tuples)

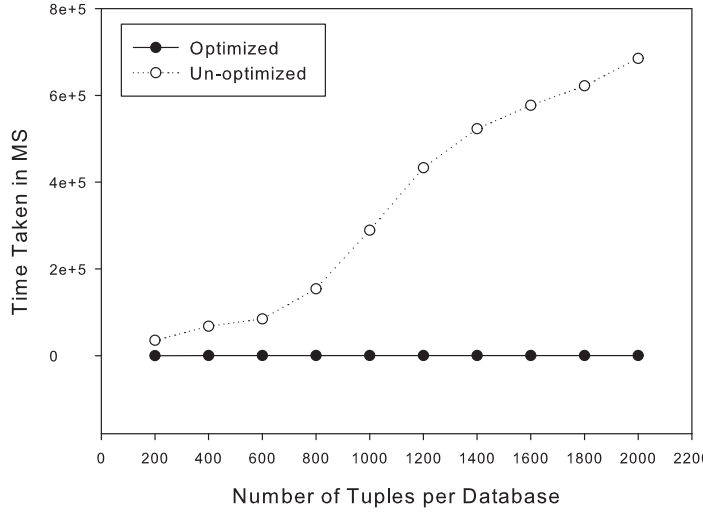

Fig. 7. Time taken to run NB-classifier on vertically partitioned data (different number of tuples in database)

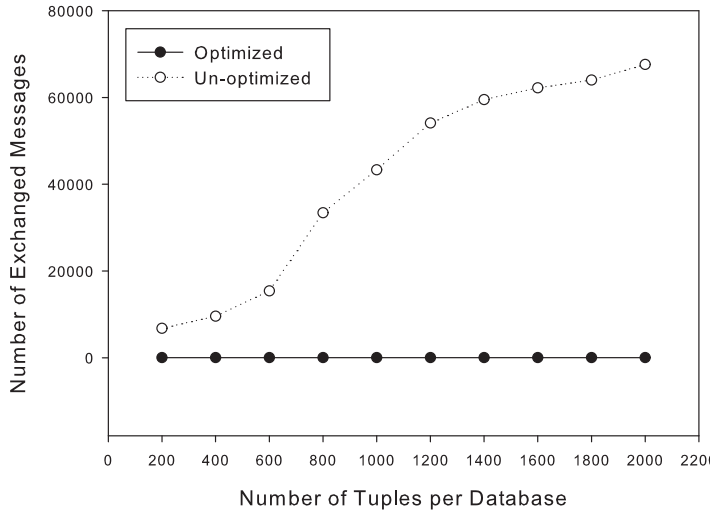

Fig. 8. Number of messages exchanged to run NB-classifier on vertically partitioned data (different number of tuples in database)

6 ADVANTAGES AND SECURITY CONSIDERATIONS

The above analysis of complexity shows that the number of messages that need to be exchanged among the sites is not dependent on the size of the database at each site. The communication complexity, in the case of vertically or horizontally partitioned data, is dependent primarily on the number and manner in which the attributes are shared among the participating sites. This is significant because it shows that as the sizes of the individual databases grow, the communication complexity of the algorithm would remain unaffected. Computational cost of local computations would grow with the database size at each individual site but our decomposable versions has an advantage in this regard also over the *transport, join, and then run NB-classifier* alternative.

If each local database D_i has p tuples, then in the worst case the join of n local databases would produce a relation containing order of p^n tuples. There is additional cost of order of $(n * p)$ comparisons for creating the *Join*. When the NB-classifier algorithm is run with this explicitly created $\mathcal{D}$, we would need to compute 3 counts. In our decomposable version, each of the n sites would be computing only 3 counts. Thus, there is tremendous saving in the computational cost when the decomposable version is executed instead of moving the data, creating a *Join* and then running the NB-classifier algorithm. Also, for the communication cost, the number of partial results that need to be transmitted is much lower than that of the messages that may have to be transmitted if entire databases are collected at some central site.

Another important gain of decomposable version is that it preserves the privacy of the data by not requiring any data tuples to be placed on a communication network. It also preserves the integrity of individual databases because no site needs to update or write into any of the participating databases. All the queries are strictly reading queries.

We have demonstrated above that NB-classifier algorithm can return same results for distributed databases without having to move the databases to a centralized site. From the point of view of data security and privacy, no data tuple is exchanged between the sites. If the information security and privacy is defined by not having to release any data tuple out of a database for transmission over the network and the reconstruction of any data tuple being impossible by the released data summaries then the above algorithm preserves the privacy of the data in each participating database. No data tuple is ever transmitted and the summaries are not sufficient to reconstruct any individual data tuple.

7 CONCLUSION

In this paper, we have presented a decomposable version of NB-classifier algorithm for vertically and horizontally partitioned datasets that are geographically distributed. We have defined the problem of learning from distributed data, presented a general strategy for transforming algorithms for learning from centralized data into algorithms for learning from distributed data. This strategy is based on the decomposition of an algorithm into information extraction and hypothesis generation components. The information extraction from distributed data entails decomposing each statistical query posed by the information extraction component of the *learner* into local computations that can be performed by the individual data sources, and a procedure for combining the results of local computations into an answer for the original query. We have applied this strategy to design algorithms for learning the NB-classifier from horizontally and vertically partitioned data.

REFERENCES

[1] Khedr, A. M.: Nearest Neighbor Clustering over Partitioned Data. Computing and Informatics, Vol. 30, 2011, pp. 1001–1026.

[2] Khedr, A. M.—Attiya, I.: Classification of Vertically Distributed Data using Directed Acyclic Graph. ICGST Conference on Computer Science and Engineering, Istanbul, Turkey 2001, pp. 207–215.

[3] Khedr, A. M.—Mahmoud, R.: Agents for Integrating Distributed Data for Function Computations. Computing and Informatics, Vol. 31, 2012, pp. 1001–1025.

[4] Khedr, A. M.—Bhatnagar, K. R.: A Decomposable Algorithm for Minimum Spanning Tree. LNCS Vol. 2198, - Distributed Computing, pp. 33-44, Springer-Verlag Heidelberg 2004.

[5] Khedr, A. M.—Bhatnagar, R. K.: Agents for Integrating Distributed Data for Complex Computations. Computing and Informatics, Vol. 26, 2007, No. 2, pp. 149–170.

[6] Khedr, A. M.: Learning k-Classifier from Distributed Databases. Computing and Informatics, Vol. 27, 2008, pp. 355–376.

[7] Khedr, A. M.—Salim, A.: Decomposable Algorithms for Finding the Nearest Pair. J. Parallel Distrib. Comput., Vol. 68, 2008, pp. 902–912.

[8] Özsu, M. T.—Valduriez, P.: Principles of Distributed Database Systems. 2nd Ed., Prentice Hall International, New Jersey, USA 1999.

[9] Langley, P.—Iba, W.—Thompson, K.: An Analysis of Bayesian classifiers. In: Proc. 10th National Conf. on Artificial Intelligence 1992, pp. 399–406.

[10] Dietterich, T. G.: Ensemble Methods in Machine Learning. Lecture Notes in Computer Science, Vol. 1857, 2000, pp. 1–15.

[11] Domingos, P.: Knowledge Acquisition from Examples Via Multiple Models. In: Proceedings of the Fourteenth International Conference on Machine Learning, Nashville, TN 1997, Morgan Kaufmann 1997, pp. 98–106.

[12] Grossman, L.—Gou, Y.: Parallel Methods for Scaling Data Mining Algorithms to Large Data Sets. In Zytkow, J. (Ed.): Handbook on Data Mining and Knowledge Discovery, Oxford University Press 2001.

[13] Prodromidis, A.—Chan, P.—Stolfo, S.: Meta-Learning in Distributed Data Mining Systems: Issues and Approaches. In: Kargupta, H., Chan, P. (Eds).: Advances of Distributed Data Mining. AAAI Press 2000.

[14] Provost, F. J.—Kolluri, V.: A Survey of Methods for Scaling up Inductive Algorithms. Data Mining and Knowledge Discovery 3, 1999, pp. 131–169.

[15] Srivastava, A.—Han, E.—Kumar, V.—Singh, V: Parallel Formulations of Decision Tree Classification Algorithms. Data Mining and Knowledge Discovery 3, 1999, pp. 237–261.

[16] Muggleton, S.: Inductive Logic Programming. Academic Press Ltd. 1992.

[17] Dzeroski, S.—Lavrac, N. eds.: Relational Data Mining. Springer-Verlag 2001.

[18] Domingos, P.—Pazzani, M.: On the Optimality of the Simple Bayesian Classifier Under Zero-One Loss. Machine Learning, Vol. 29, 1997, pp. 103–130.

[19] Zhang, H.—Ling, C. X.: Geometric Properties of Naive Bayes in Nominal Domains. European Conference on Machine Learning 2001, pp. 588–599.

[20] Garg, A.—Roth, D.: Understanding Probabilistic Classifiers. European Conference on Machine Learning, Lecture Notes in Artificial Intelligence 2001, pp. 179–191.

[21] Kantarcioglu, M.—Vaidya, J.: Privacy Preserving Naive Bayes Classifier for Horizontally Partitioned Data. In IEEE Workshop on Privacy Preserving Data Mining 2003, pp. 3–9.

[22] Vaidya, J.—Clifton, C.: Privacy Preserving Naive Bayes Classifier for Vertically Partitioned Data. In IEEE Workshop on Privacy Preserving Data Mining 2003, pp. 3–9.

[23] Kersting, K.—de Raedt, L.: Bayesian Logic Programs. In F. Furukawa, S. Muggleton, D. Michie, and L. de Raedt (Eds.): Proceedings of the Seventeenth Machine Intelligence Workshop, Bury St. Edmunds, Suffolk, UK 2000.

[24] McCallum, A.—Nigam, K.: A Comparison of Event Models for Naive Bayes Text Classification. In AAAI-98 – Workshop on Learning for Text Categorization.

[25] Mitchell, T. M.: Machine Learning. McGraw Hill 1997.

[26] Wang, C.—Chen, M.: On the Complexity of Distributed Query Optimization. IEEE Transactions on Knowledge and Data Engineering, Vol. 8, 1996, No. 4, pp. 650–662.

Ahmed M. Khedr received his B. Sc. degree in mathematics in June 1989 and his M. Sc. degree in optimal control in July 1995, both from Zagazig University, Egypt. In July 1999 he received his M. Sc. and in March 2003 his Ph. D. degree, both in computer science and Engineering, from University of Cincinnati, Ohio, USA. From March 2003 to January 2004, he was a research Assistant Professor at ECECS Department of University of Cincinnati, USA. From January 2004 till May 2009, he worked as Assistant Professor at Zagazig University, Egypt; from September 2009 till September 2010 he worked as Associate Professor at the Department of Computer Science of College of Computers and Information Systems, Taif University, Saudi Arabia, and from September 2010 till now he is working as Associate Professor at the Department of Computer Sciences, College of Science, Sharjah University, UAE. In June 2009, he was awarded by the State Prize of Distinction in Advanced Technology. He has co-authored 42 works in journals and conferences related to optimal control, wireless swensor networks, distributed computing, and bioinformatics.